# Lesson: Classes in Python

## Learning Objectives

1. Understand the concept of classes and objects.
2. Learn how to create and use classes in Python.
3. Understand key components of a class: attributes and methods.
4. Differentiate between `static` and `instance`.
5. Master advanced concepts: inheritance and polymorphism.

---

## Part 1: Introduction to Classes and Objects

- **Class:** A blueprint or template for creating objects.
- **Object:** A specific instance created from a class.

### Example:

```
# Creating a simple class
class Dog:
    # Attribute
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    # Method
    def bark(self):
        print(f"{self.name} (a {self.breed}) is barking: Woof Woof!")

# Creating an object
dog1 = Dog("Bobby", "Golden Retriever")
dog1.bark()
```

**Output:**

```
Bobby (a Golden Retriever) is barking: Woof Woof!
```

---

## Part 2: Key Components of a Class

### 1. Attributes

- Attributes are defined in the `__init__` method.
- **Instance Attribute:** Belongs to a specific object.
- **Class Attribute:** Shared among all objects of the class.

### Example:

```
class Cat:
    # Class Attribute
    species = "Cat"

    def __init__(self, name, color):
        # Instance Attributes
        self.name = name
        self.color = color

# Creating objects
cat1 = Cat("Mimi", "White")
cat2 = Cat("Tom", "Grey")

print(cat1.species)  # Output: Cat
print(cat1.name)     # Output: Mimi
print(cat2.color)    # Output: Grey
```

---

### 2. Methods

- **Instance Method:** Operates on a specific object, uses the `self` keyword.
- **Static Method:** Does not rely on any object, uses the `@staticmethod` decorator.
- **Class Method:** Works on the class itself, uses the `@classmethod` decorator.

### Example:

```
class Math:
    # Static Method
    @staticmethod
    def add(a, b):
        return a + b

    # Class Method
    @classmethod
    def description(cls):
        return "This is the Math class"

# Calling Static Method
print(Math.add(3, 5))  # Output: 8

# Calling Class Method
print(Math.description())  # Output: This is the Math class
```

---

## Part 3: Differentiating Static, Instance, and Class Methods

| Type | Uses `self`? | Uses `cls`? | Called by |
|---|---|---|---|
| Instance | Yes | No | Object |
| Static | No | No | Class |
| Class | No | Yes | Class |

---

## Part 4: Inheritance

- **Inheritance:** A child class inherits attributes and methods from a parent class.
- **Keyword:** `super()` is used to call methods of the parent class.

### Example:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} is making a sound.")

# Child class inheriting from parent class
class Dog(Animal):
    def speak(self):
        print(f"{self.name} is barking: Woof Woof!")

dog = Dog("Bobby")
dog.speak()  # Output: Bobby is barking: Woof Woof!
```

---

## Part 5: Polymorphism

- Polymorphism allows the same method to have different behaviors depending on the class.

### Example:

```
class Bird:
    def fly(self):
        print("Birds fly in the sky.")

class Penguin(Bird):
    def fly(self):
        print("Penguins cannot fly!")

# Polymorphism
def describe_flight(bird):
    bird.fly()

bird1 = Bird()
penguin = Penguin()

describe_flight(bird1)    # Output: Birds fly in the sky.
describe_flight(penguin)  # Output: Penguins cannot fly!
```

---

## Part 6: Practical Exercises

### Exercise 1: Create a Book Class

**Description:** Create a `Book` class with attributes `title`, `author`, `price`. Create a method `show_info()` to print the book's details.

### Exercise 2: Student Management

**Description:** Create a `Student` class with attributes `name`, `age`, `grades`. Write a method to calculate the student's average grade.

### Exercise 3: Inheritance

**Description:** Create a `Vehicle` class with attributes `name` and method `move()`. Create a `Car` class that inherits from `Vehicle` and overrides the `move()` method.

### Exercise 4: Polymorphism

**Description:** Create two classes `Dog` and `Cat`, each with a `speak()` method. Write a common function to call `speak()` for these objects.

---

## Summary

- Classes and objects are the core of Object-Oriented Programming (OOP).
- Concepts like inheritance and polymorphism make code more flexible and maintainable.
- Practice the exercises to strengthen your understanding of classes in Python.